Feature Translation

More Regression and Classification approaches

Linear Regression

- Given data with n dimensional variables and 1 target-variable (real number)

  $$\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_m, y_m)\}$$

  Where $\mathbf{x} \in \Re^n, y \in \Re$
- The objective: Find a function f that returns the best fit. $f : \Re^n \to \Re$
- Assume that the relationship between X and y is approximately linear. The model can be represented as (w represents coefficients and b is an intercept)

  $$f(w_1, \ldots, w_n, b) = y = \mathbf{w} \cdot \mathbf{x} + b + \varepsilon$$

Linear Regression

- To find the best fit, we minimize the sum of squared errors → Least square estimation

$$\min \sum_{i=1}^{m} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{m} (y_i - (\mathbf{w} \cdot \mathbf{x}_i + b))^2$$

- The solution can be found by solving

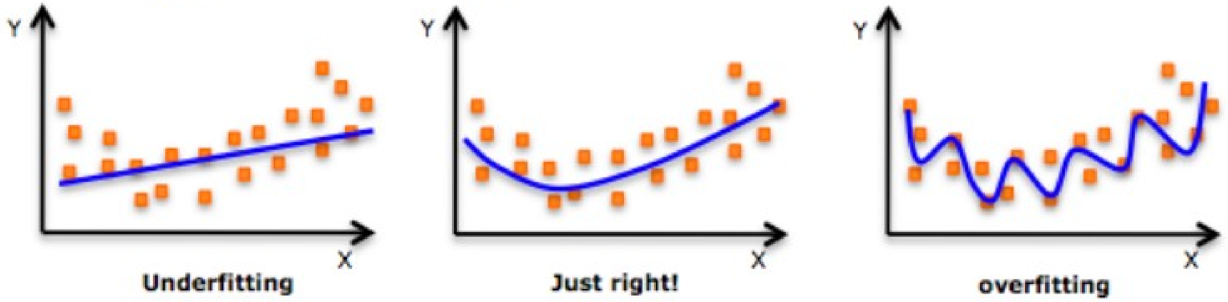
  $\hat{\mathbf{w}} = (X^T X)^{-1} X^T Y$

  (By taking the derivative of the above objective function w.r.t. $\mathbf{w}$)

- In MATLAB, the back-slash operator computes a least square solution.

Linear Regression

$$\min \sum_{i=1}^{m} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{m} (y_i - (\hat{\mathbf{w}} \cdot \mathbf{x}_i + \hat{b}))^2$$



- To ovoid over-fitting, a regularization term can be introduced (minimize a magnitude of w)
  - LASSO: $$\min \sum_{i=1}^{m} (y_i - \mathbf{w} \cdot \mathbf{x}_i - b)^2 + C \sum_{j=1}^{n} | w_j |$$
  - Ridge regression: $$\min \sum_{i=1}^{m} (y_i - \mathbf{w} \cdot \mathbf{x}_i - b)^2 + C \sum_{j=1}^{n} | \mathbf{w}_j^2 |$$

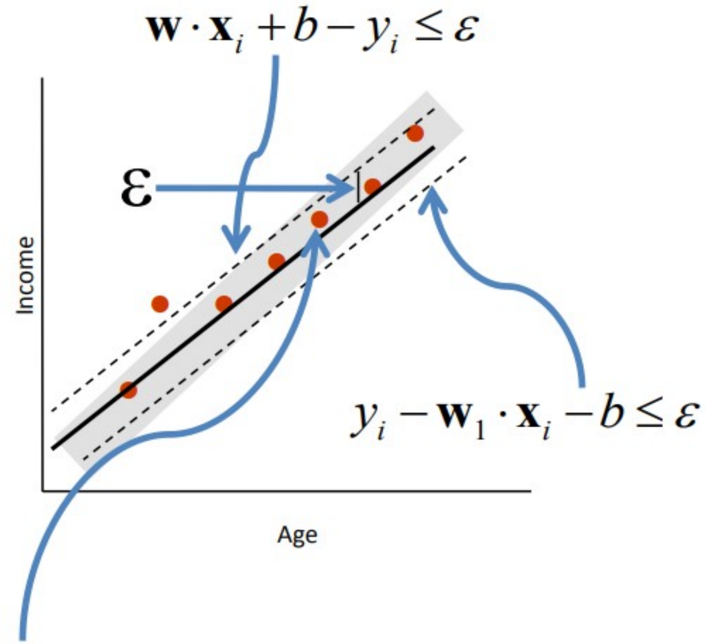
Support Vector Regression

- Find a function, f(x), with at most ε-deviation from the target y

The problem can be written as a convex optimization problem

$$\min \frac{1}{2} \| \mathbf{w} \|^2$$
$$s.t.\ y_i - \mathbf{w}_1 \cdot \mathbf{x}_i - b \le \varepsilon;$$
$$\mathbf{w}_1 \cdot \mathbf{x}_i + b - y_i \le \varepsilon;$$

C: trade off the complexity



What if the problem is not feasible?

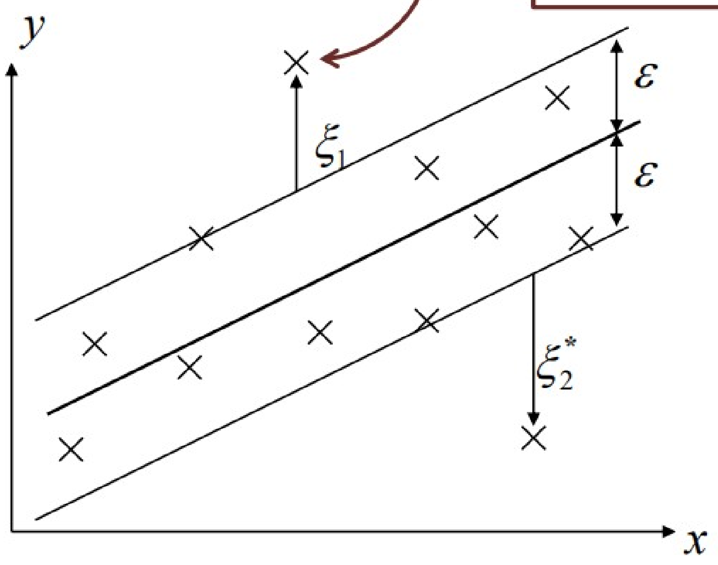
We can introduce slack variables (similar to soft margin loss function).

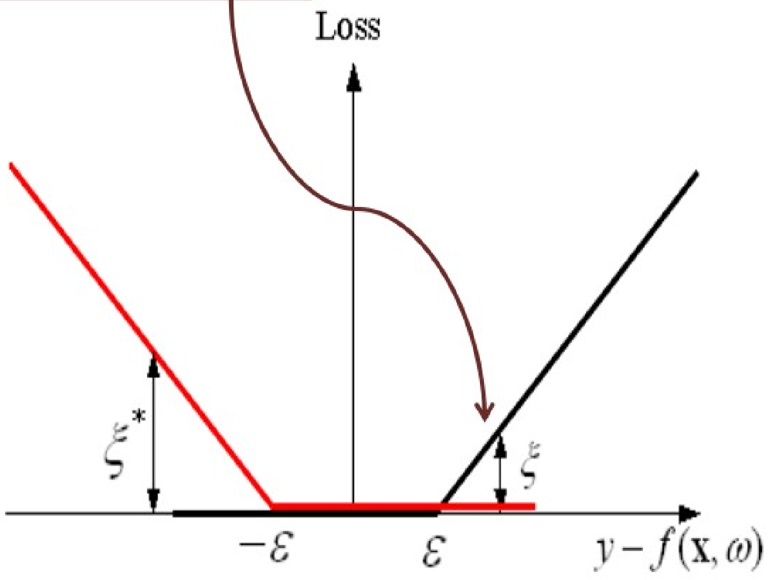
We do not care about errors as long as they are less than ε

Support Vector Regression

Assume linear parameterization $f(\mathbf{x},\omega) = \mathbf{w}\cdot\mathbf{x} + b$

Only the point outside the ε-region contribute to the final cost



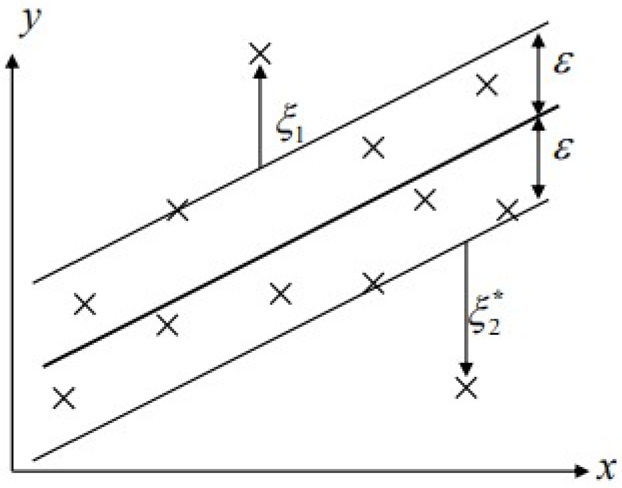


$$L_\varepsilon(y, f(\mathbf{x},\omega)) = \max\left(|\, y - f(\mathbf{x},\omega)\,| - \varepsilon, 0\right)$$

Soft margin

Given training data

$$(\mathbf{x}_i, y_i) \quad i = 1, \ldots, m$$



Minimize

$$\frac{1}{2} \| \mathbf{w} \|^2 + C \sum_{i=1}^{m} (\xi_i + \xi_i^*)$$

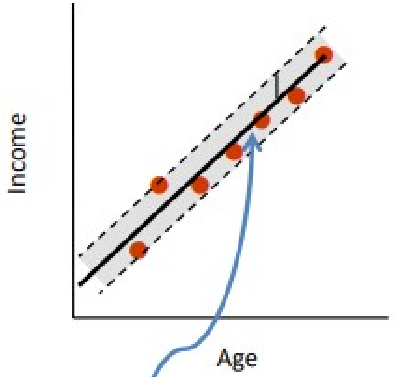
Under constraints

$$\begin{cases} y_i - (\mathbf{w} \cdot \mathbf{x}_i) - b \leq \varepsilon + \xi_i \\ (\mathbf{w} \cdot \mathbf{x}_i) + b - y_i \leq \varepsilon + \xi_i^* \\ \xi_i \, , \xi_i^* \geq 0, i = 1, \ldots, m \end{cases}$$

Linear versus Non-linear SVR

- **Linear case**

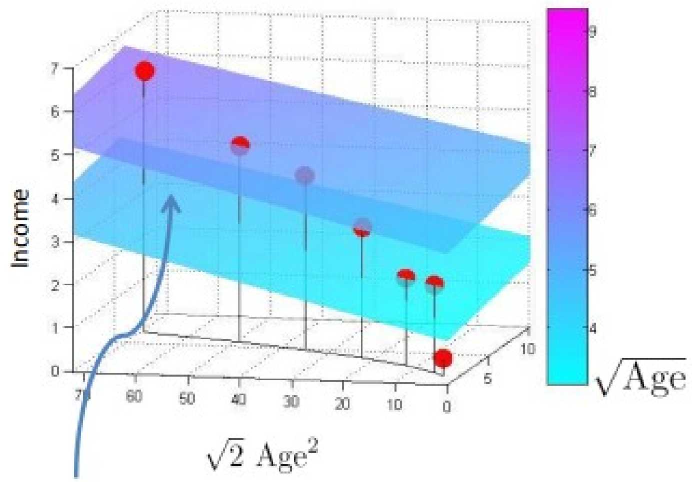
$$f : age \rightarrow income$$



$$y_i = \mathbf{w}_1 \cdot \mathbf{x}_i + b$$

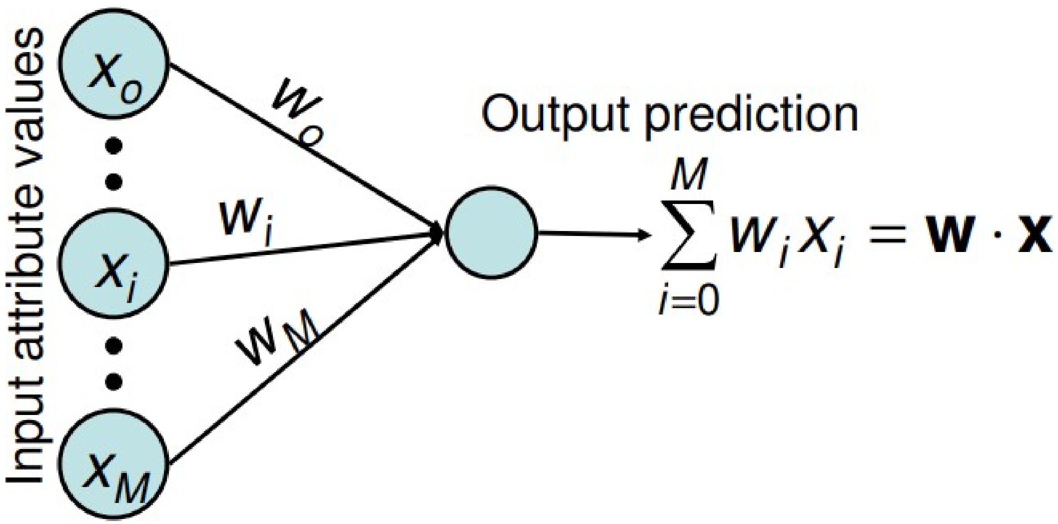
- **Non-linear case**
  - Map data into a higher dimensional space, e.g.,

$$f : (\sqrt{age}, \sqrt{2}age^2) \rightarrow income$$



$$y_i = \mathbf{w}_1 \sqrt{\mathbf{x}_i} + \mathbf{w}_2 \sqrt{2}\mathbf{x}_i^2 + b$$

Regression: Neural Networks



Regression: Neural Networks

Perceptron Training

- Given input training data x^k with corresponding value y^k

1. Compute error:

$$\boxed{\delta_k \leftarrow y^k - \mathbf{w} \cdot \mathbf{x}^k}$$

2. Update NN weights:

$$\boxed{w_i \leftarrow w_i + \alpha \delta_k x_i^k}$$

Regression: Neural Networks

- Update has many names: delta rule, gradient rule, LMS rule
- Update is guaranteed to converge to the best linear fit (global minimum of E)
- Of course, there are more direct ways of solving the linear regression problem by using linear algebra techniques. It boils down to a simple matrix inversion (not shown here).
- In fact, the perceptron training algorithm can be much, much slower than the direct solution
- Use of MLP with Backpropagation for Non-Linear Regression

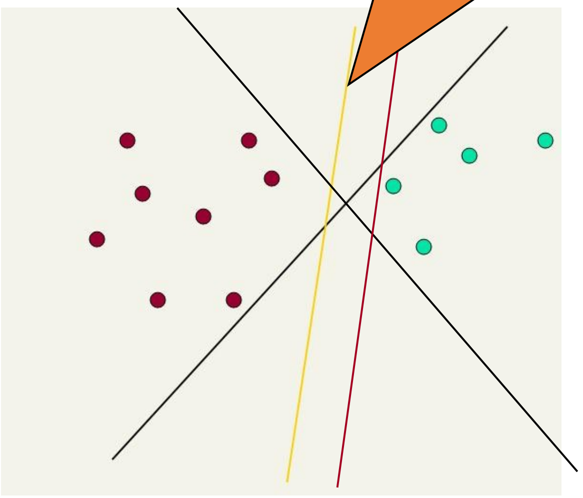
Perceptron Learning Algorithm

- First neural network learning model in the 1960's
- Simple and limited (single layer model)
- Basic concepts are similar for multi-layer models, so this is a good learning tool
- Still used in some current applications (large business problems, where intelligibility is needed, etc.)

Linear classifiers: Which Hyperplane?

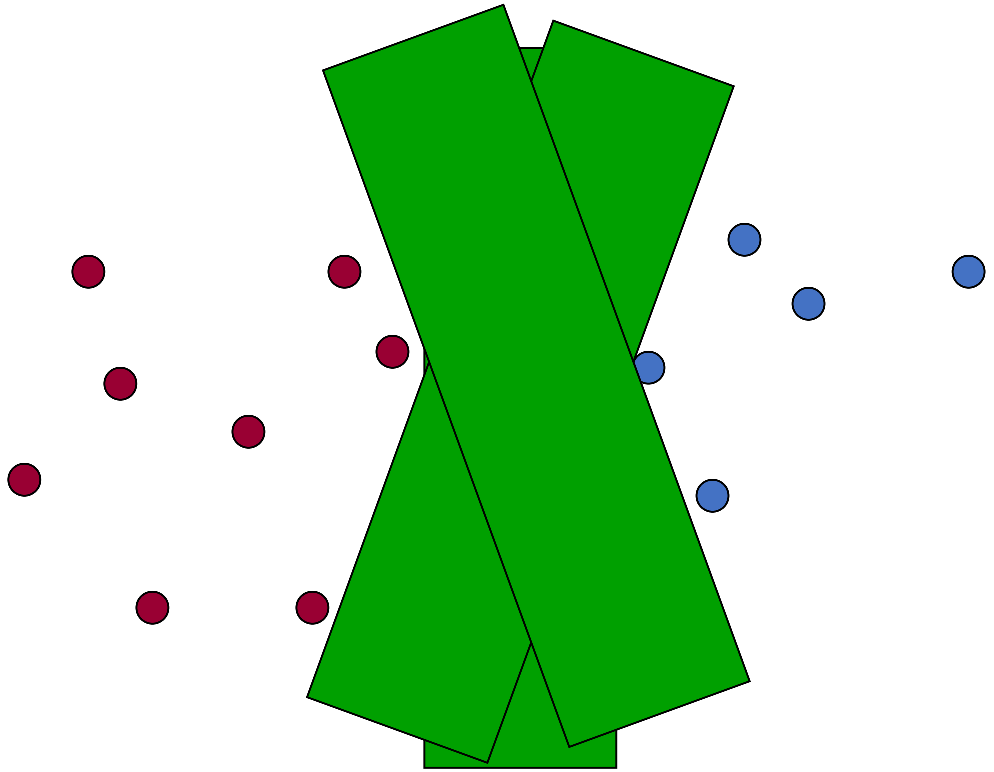
- Lots of possible solutions for *a, b, c.*
- Some methods find a separating hyperplane, but not the optimal one [according to some criterion of expected goodness]
  - E.g., perceptron
- Support Vector Machine (SVM) finds an optimal* solution.
  - Maximizes the distance between the hyperplane and the “difficult points” close to decision boundary
  - One intuition: if there are no points near the decision surface, then there are no very uncertain classification decisions

This line represents the decision boundary:
$ax + by - c = 0$



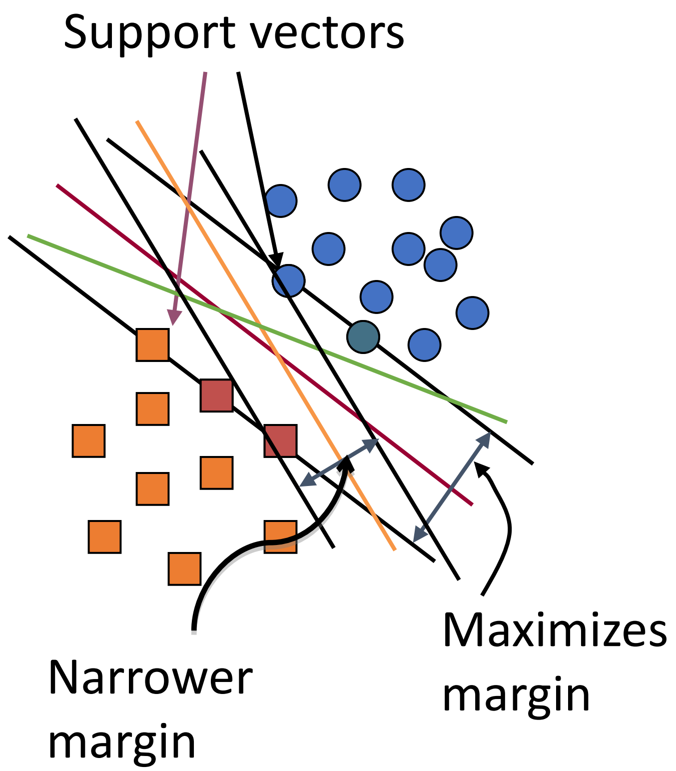
Another intuition

- If you have to place a fat separator between classes, you have less choices, and so the capacity of the model has been decreased



Support Vector Machine (SVM)

- SVMs maximize the *margin* around the separating hyperplane.
  - A.k.a. large margin classifiers
- The decision function is fully specified by a subset of training samples, *the support vectors*.
- Solving SVMs is a *quadratic programming* problem
- Seen by many as the most successful current text classification method*



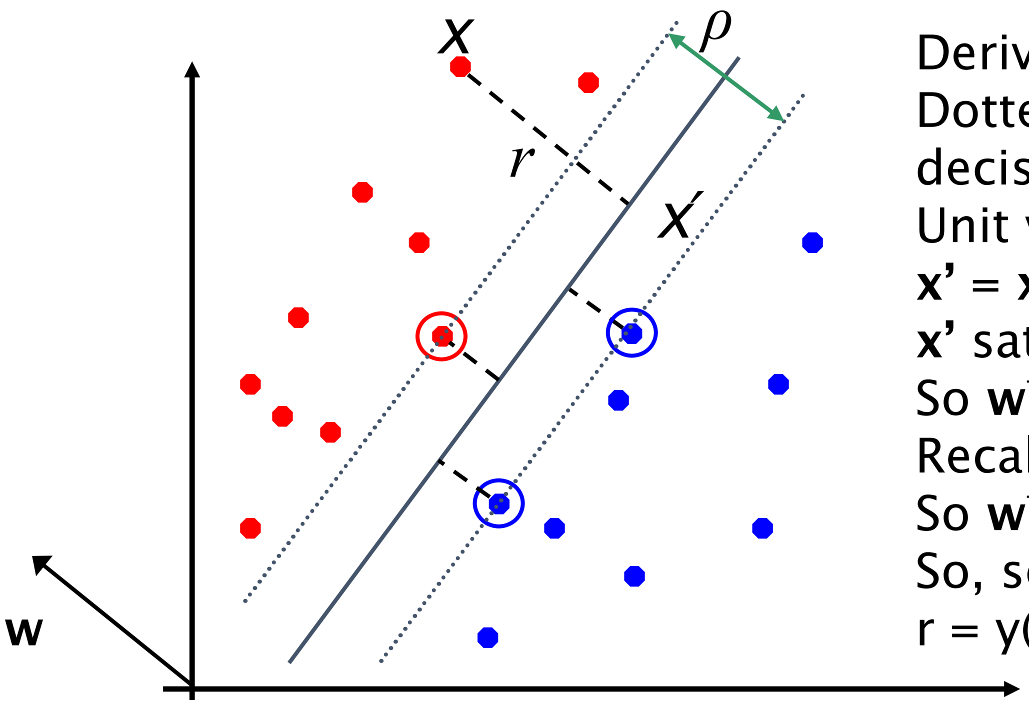
*but other discriminative methods often perform very similarly

Maximum Margin: Formalization

- **w**: decision hyperplane normal vector
- $\mathbf{x}_i$: data point *i*
- y_i: class of data point *i* (+1 or -1) NB: Not 1/0
- Classifier is: $f(\mathbf{x}_i) = \text{sign}(\mathbf{w}^T\mathbf{x}_i + b)$
- Functional margin of $\mathbf{x}_i$ is: $y_i\,(\mathbf{w}^T\mathbf{x}_i + b)$
  - But note that we can increase this margin simply by scaling **w**, **b**….
- Functional margin of dataset is twice the minimum functional margin for any point
  - The factor of 2 comes from measuring the whole width of the margin

Geometric Margin

- Distance from example to the separator is $r = y\frac{\mathbf{w}^T\mathbf{x}+b}{\|\mathbf{w}\|}$
- Examples closest to the hyperplane are ***support vectors***.
- ***Margin*** ρ of the separator is the width of separation between support vectors of classes.



Derivation of finding *r:*
Dotted line **x'−x** is perpendicular to decision boundary so parallel to **w**.
Unit vector is **w**/|**w**|, so line is r**w**/|**w**|.
x' = **x** – yr**w**/|**w**|.
x' satisfies $\mathbf{w}^T\mathbf{x}'$+b = 0.
So $\mathbf{w}^T$(**x** –yr**w**/|**w**|) + b = 0
Recall that |**w**| = sqrt($\mathbf{w}^T\mathbf{w}$).
So $\mathbf{w}^T\mathbf{x}$ –yr|**w**| + b = 0
So, solving for r gives:
r = y($\mathbf{w}^T\mathbf{x}$ + b)/|**w**|

Linear SVM Mathematically

The linearly separable case

- Assume that all data is at least distance 1 from the hyperplane, then the following two constraints follow for a training set $\{(\mathbf{x_i}, y_i)\}$

$$\mathbf{w^T x_i} + b \geq 1 \quad \text{if } y_i = 1$$

$$\mathbf{w^T x_i} + b \leq -1 \quad \text{if } y_i = -1$$

- For support vectors, the inequality becomes an equality
- Then, since each example's distance from the hyperplane is

$$r = y\frac{\mathbf{w}^T\mathbf{x} + b}{\|\mathbf{w}\|}$$

- The margin is:

$$\rho = \frac{2}{\|\mathbf{w}\|}$$

Linear Support Vector Machine (SVM)

- **Hyperplane**

  $\mathbf{w}^T \mathbf{x} + b = 0$

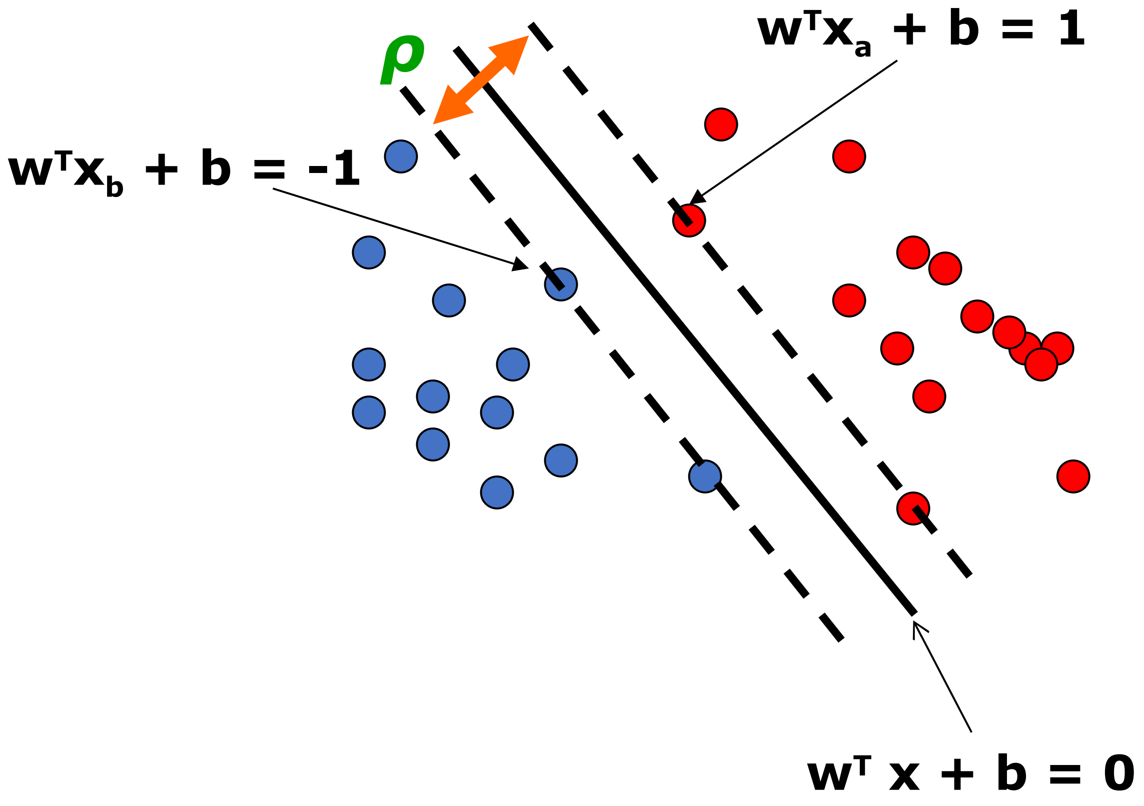
- **Extra scale constraint**:

  $\min_{i=1,\ldots,n} |\mathbf{w}^T\mathbf{x}_i + \mathbf{b}| = 1$

- This implies:

  $w^T(x_a - x_b) = 2$

  $\rho = ||x_a - x_b||_2 = 2/||\mathbf{w}||_2$



Linear SVMs Mathematically (cont.)

- Then we can formulate the *quadratic optimization problem:*

> Find **w** and *b* such that
>
> $\rho = \frac{2}{\|\mathbf{w}\|}$ is maximized; and for all $\{(\mathbf{x_i}, y_i)\}$
>
> $\mathbf{w}^T\mathbf{x_i} + b \geq 1$ if $y_i = 1$; $\mathbf{w}^T\mathbf{x_i} + b \leq -1$ if $y_i = -1$

- A better formulation (min **||w||** = max 1/ **||w||**):

> Find **w** and *b* such that
>
> $\mathbf{\Phi}(\mathbf{w}) = \frac{1}{2}\,\mathbf{w}^T\mathbf{w}$ is minimized;
>
> and for all $\{(\mathbf{x_i}, y_i)\}$: $y_i(\mathbf{w}^T\mathbf{x_i} + b) \geq 1$

Solving the Optimization Problem

Find $\mathbf{w}$ and b such that
$\boldsymbol{\Phi}(\mathbf{w}) = ½\ \mathbf{w}^T\mathbf{w}$ is minimized;
and for all $\{(\mathbf{x_i}, y_i)\}$: $y_i(\mathbf{w}^T\mathbf{x_i} + b) \geq 1$

- This is now optimizing a *quadratic* function subject to *linear* constraints
- Quadratic optimization problems are a well-known class of mathematical programming problem, and many (intricate) algorithms exist for solving them (with many special ones built for SVMs)
- The solution involves constructing a *dual problem* where a *Lagrange multiplier* α_i is associated with every constraint in the primary problem:

Find $\alpha_1 \ldots \alpha_N$ such that
$\mathbf{Q}(\boldsymbol{\alpha}) = \Sigma\alpha_i - ½\Sigma\Sigma\alpha_i\alpha_j y_i y_j \mathbf{x_i}^T\mathbf{x_j}$ is maximized and
(1) $\Sigma\alpha_i y_i = 0$
(2) $\alpha_i \geq 0$ for all α_i

The Optimization Problem Solution

- The solution has the form:

$$\boxed{\mathbf{w} = \Sigma\alpha_i y_i \mathbf{x_i} \qquad b = y_k - \mathbf{w^T x_k} \text{ for any } \mathbf{x_k} \text{ such that } \alpha_k \neq 0}$$

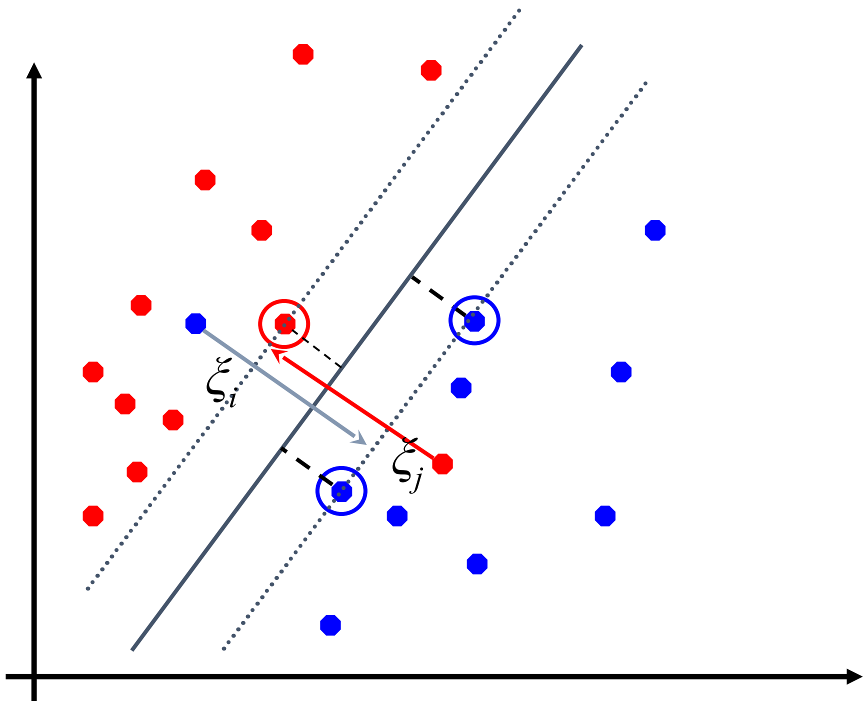
- Each non-zero α_i indicates that corresponding $\mathbf{x_i}$ is a support vector.
- Then the classifying function will have the form:

$$\boxed{f(\mathbf{x}) = \Sigma\alpha_i y_i \mathbf{x_i^T x} + b}$$

- Notice that it relies on an *inner product* between the test point $\mathbf{x}$ and the support vectors $\mathbf{x_i}$
  - We will return to this later.
- Also keep in mind that solving the optimization problem involved computing the inner products $\mathbf{x_i^T x_j}$ between all pairs of training points.

Soft Margin Classification

- If the training data is not linearly separable, *slack variables* ξ_i can be added to allow misclassification of difficult or noisy examples.
- Allow some errors
  - Let some points be moved to where they belong, at a cost
- Still, try to minimize training set errors, and to place hyperplane "far" from each class (large margin)



Soft Margin Classification Mathematically

- The old formulation:

> Find $\mathbf{w}$ and b such that
> $\mathbf{\Phi}(\mathbf{w}) = ½\ \mathbf{w}^{\mathrm{T}}\mathbf{w}$ is minimized and for all $\{(\mathbf{x_i}, y_i)\}$
> $y_i\ (\mathbf{w^T x_i} + \mathrm{b}) \geq 1$

- The new formulation incorporating slack variables:

> Find $\mathbf{w}$ and b such that
> $\mathbf{\Phi}(\mathbf{w}) = ½\ \mathbf{w}^{\mathrm{T}}\mathbf{w} + C\Sigma\xi_i$ is minimized and for all $\{(\mathbf{x_i}, y_i)\}$
> $y_i\ (\mathbf{w^T x_i} + b) \geq 1 - \xi_i$ and $\xi_i \geq 0$ for all i

- Parameter *C* can be viewed as a way to control overfitting
  - A regularization term

Soft Margin Classification – Solution

- The dual problem for soft margin classification:

> Find $\alpha_1 \ldots \alpha_N$ such that
>
> $\mathbf{Q}(\boldsymbol{\alpha}) = \Sigma\alpha_i - \frac{1}{2}\Sigma\Sigma\alpha_i\alpha_j y_i y_j \mathbf{x_i}^T\mathbf{x_j}$ is maximized and
>
> (1) $\Sigma\alpha_i y_i = 0$
>
> (2) $0 \leq \alpha_i \leq C$ for all α_i

- Neither slack variables ξ_i nor their Lagrange multipliers appear in the dual problem!
- Again, $\mathbf{x_i}$ with non-zero α_i will be support vectors.
- Solution to the dual problem is:

> $\mathbf{w} = \Sigma\alpha_i y_i \mathbf{x_i}$
>
> $b = y_k(1 - \xi_k) - \mathbf{w}^T\mathbf{x}_k$ where $k = \underset{k'}{\operatorname{argmax}}\ \alpha_{k'}$

w is not needed explicitly for classification!

> $f(\mathbf{x}) = \Sigma\alpha_i y_i \mathbf{x_i}^T\mathbf{x} + b$

Classification with SVMs

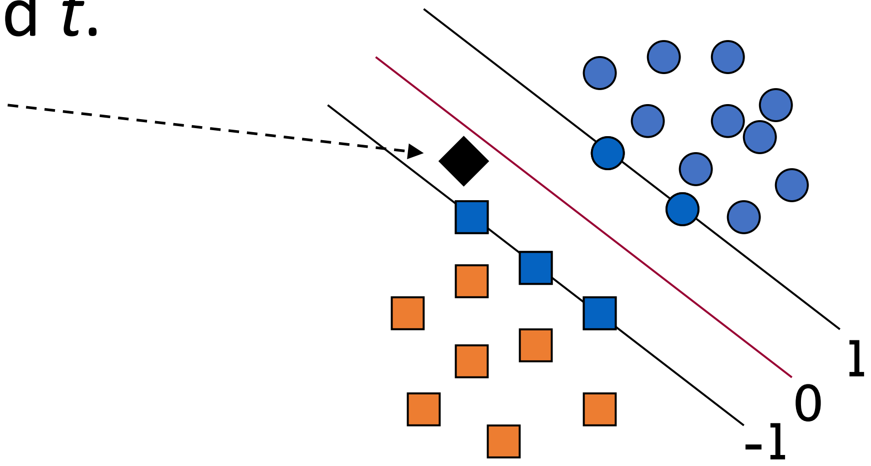
- Given a new point **x**, we can score its projection onto the hyperplane normal:
  - I.e., compute score: $\mathbf{w}^T\mathbf{x} + b = \Sigma\alpha_i y_i \mathbf{x}_i^T\mathbf{x} + b$
    - Decide class based on whether < or > 0

  - Can set confidence threshold *t*.

Score > *t*: yes

Score < -*t*: no

Else: don’t know



Linear SVMs: Summary

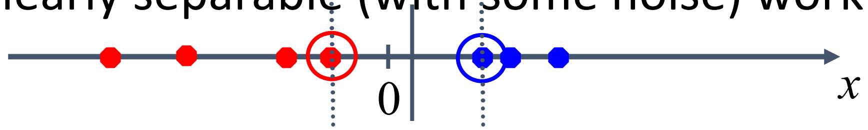
- The classifier is a *separating hyperplane.*
- The most "important" training points are the support vectors; they define the hyperplane.
- Quadratic optimization algorithms can identify which training points $\mathbf{x}_i$ are support vectors with non-zero Lagrangian multipliers α_i.
- Both in the dual formulation of the problem and in the solution, training points appear only inside inner products:

Find $\alpha_1 \ldots \alpha_N$ such that
$\mathbf{Q}(\boldsymbol{\alpha}) = \Sigma\alpha_i - ½\Sigma\Sigma\alpha_i\alpha_j y_i y_j \boxed{\mathbf{x_i}^T\mathbf{x_j}}$ is maximized and
(1) $\Sigma\alpha_i y_i = 0$
(2) $0 \leq \alpha_i \leq C$ for all α_i

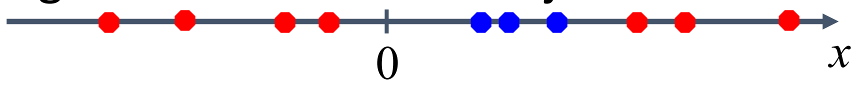
$$f(\mathbf{x}) = \Sigma\alpha_i y_i \boxed{\mathbf{x_i}^T\mathbf{x}} + b$$

Non-linear SVMs

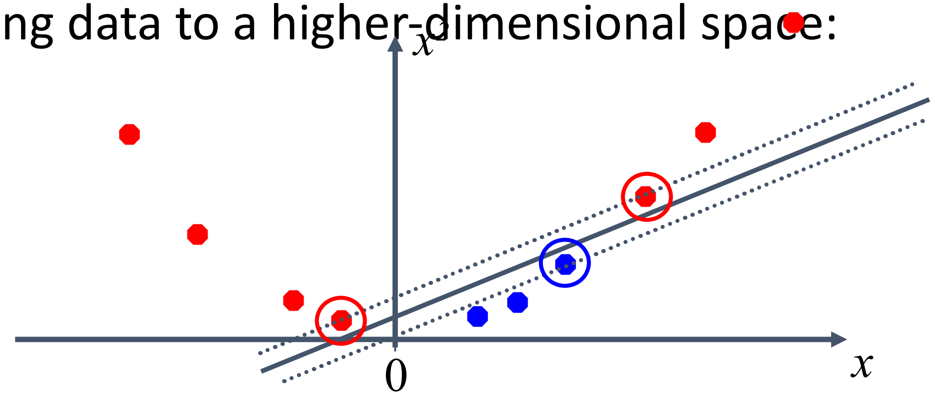
- Datasets that are linearly separable (with some noise) work out great:



- But what are we going to do if the dataset is just too hard?

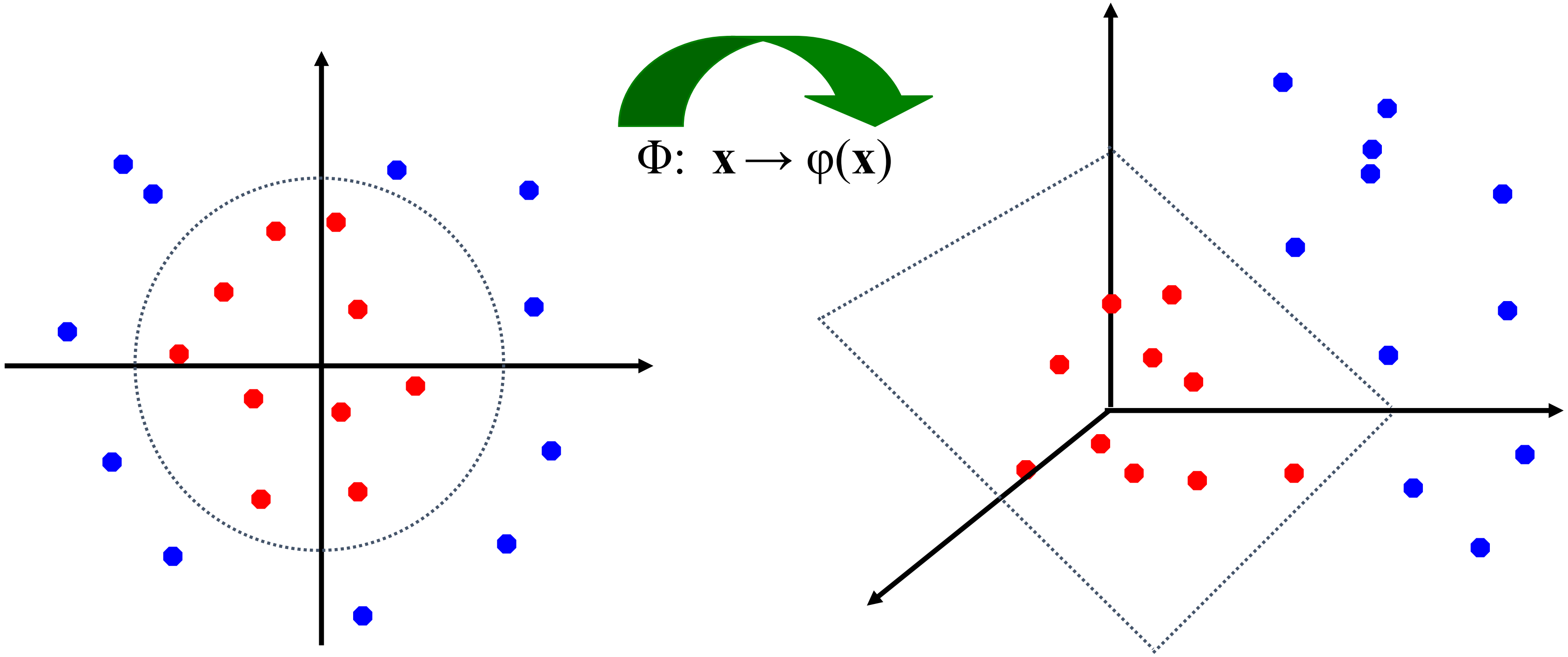


- How about … mapping data to a higher-dimensional space:



Non-linear SVMs: Feature spaces

- General idea: the original feature space can always be mapped to some higher-dimensional feature space where the training set is separable:



The “Kernel Trick”

- The linear classifier relies on an inner product between vectors $K(\mathbf{x}_i,\mathbf{x}_j)=\mathbf{x}_i^T\mathbf{x}_j$
- If every datapoint is mapped into high-dimensional space via some transformation Φ: $\mathbf{x} \rightarrow \phi(\mathbf{x})$, the inner product becomes:

$$K(\mathbf{x}_i,\mathbf{x}_j)= \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)$$

- A *kernel function* is some function that corresponds to an inner product in some expanded feature space.
- Example:

  2-dimensional vectors $\mathbf{x}=[x_1\ \ x_2]$; let $K(\mathbf{x}_i,\mathbf{x}_j)=(1 + \mathbf{x}_i^T\mathbf{x}_j)^2$,

  Need to show that $K(\mathbf{x}_i,\mathbf{x}_j)= \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)$:

  $K(\mathbf{x}_i,\mathbf{x}_j)=(1 + \mathbf{x}_i^T\mathbf{x}_j)^2 = 1+ x_{i1}^2x_{j1}^2 + 2\, x_{i1}x_{j1}\, x_{i2}x_{j2} + x_{i2}^2x_{j2}^2 + 2x_{i1}x_{j1} + 2x_{i2}x_{j2} =$

  $= [1\ \ x_{i1}^2\ \ \sqrt{2}\, x_{i1}x_{i2}\ \ x_{i2}^2\ \ \sqrt{2}x_{i1}\ \ \sqrt{2}x_{i2}]^T\, [1\ \ x_{j1}^2\ \ \sqrt{2}\, x_{j1}x_{j2}\ \ x_{j2}^2\ \ \sqrt{2}x_{j1}\ \ \sqrt{2}x_{j2}]$

  $= \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)$ where $\phi(\mathbf{x}) = [1\ \ x_1^2\ \ \sqrt{2}\, x_1x_2\ \ x_2^2\ \ \sqrt{2}x_1\ \ \sqrt{2}x_2]$

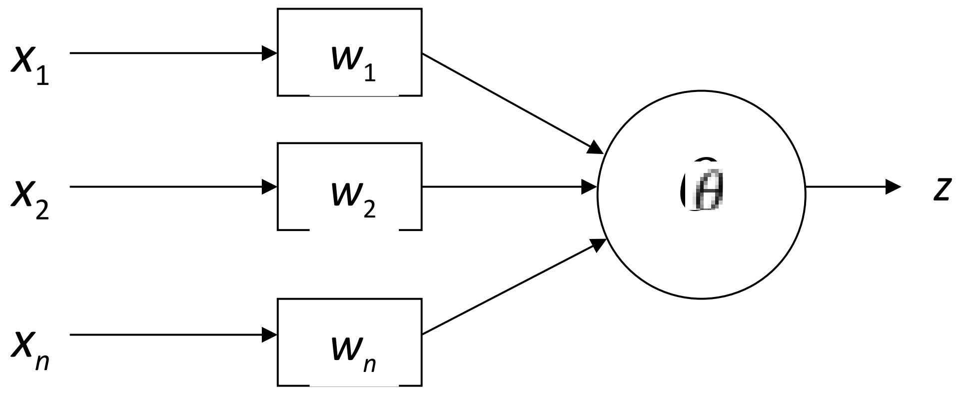
Kernels

- Why use kernels?
  - Make non-separable problem separable.
  - Map data into better representational space
- Common kernels
  - Linear
  - Polynomial **$K(x,z) = (1+x^Tz)^d$**
    - Gives feature conjunctions
  - Radial basis function (infinite dimensional space)

$$K(\mathbf{x}_i, \mathbf{x}_j) = e^{-\|\mathbf{x}_i - \mathbf{x}_j\|^2 / 2\sigma^2}$$

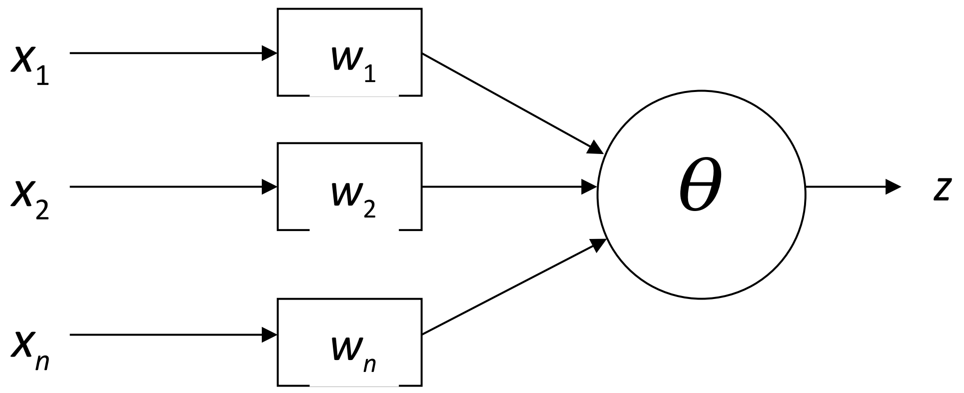
- Haven’t been very useful in text classification

Classification: Perceptron Node – Threshold Logic Unit



$$z = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} x_i w_i \geq \theta \\ 0 & \text{if } \sum_{i=1}^{n} x_i w_i < \theta \end{cases}$$

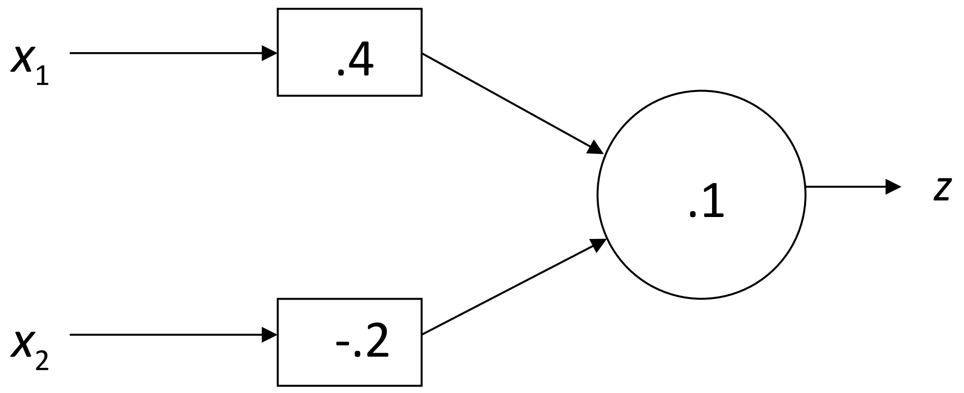
Perceptron Node – Threshold Logic Unit



- Learn weights such that an objective function is maximized.
- What objective function should we use?
- What learning algorithm should we use?

$$z = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} x_i w_i \geq \theta \\ 0 & \text{if } \sum_{i=1}^{n} x_i w_i < \theta \end{cases}$$

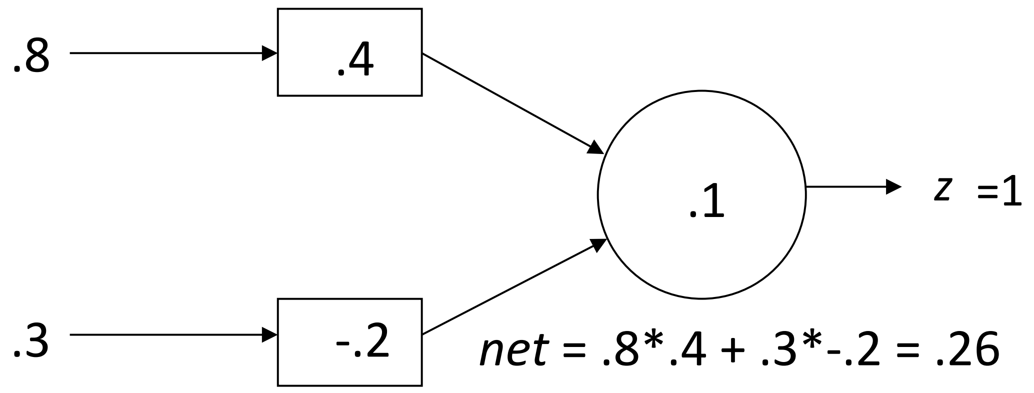
Perceptron Learning Algorithm



x_1	x_2	t
.8	.3	1
.4	.1	0

$$z = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} x_i w_i \geq \theta \\ 0 & \text{if } \sum_{i=1}^{n} x_i w_i < \theta \end{cases}$$

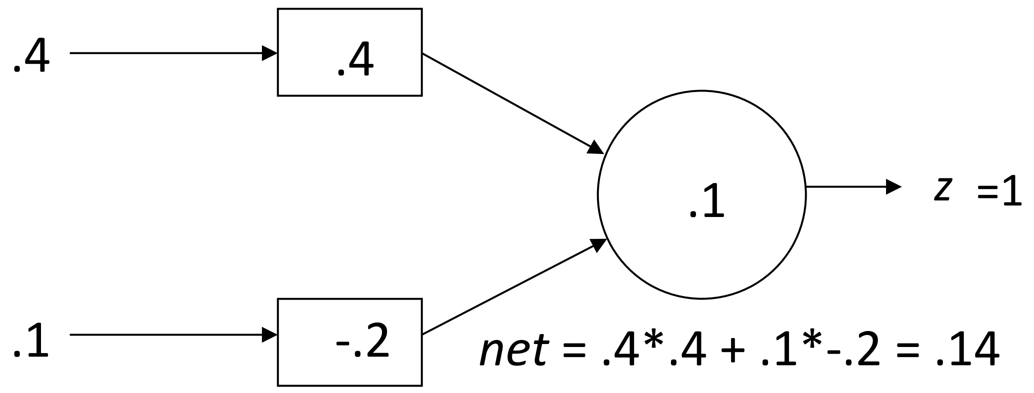
First Training Instance



x_1	x_2	t
.8	.3	1
.4	.1	0

$$z = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} x_i w_i \geq \theta \\ 0 & \text{if } \sum_{i=1}^{n} x_i w_i < \theta \end{cases}$$

Second Training Instance



x_1	x_2	t
.8	.3	1
.4	.1	0

$$z = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} x_i w_i \geq \theta \\ 0 & \text{if } \sum_{i=1}^{n} x_i w_i < \theta \end{cases}$$

$$\Delta w_i = (t - z) * c * x_i$$

Perceptron Rule Learning

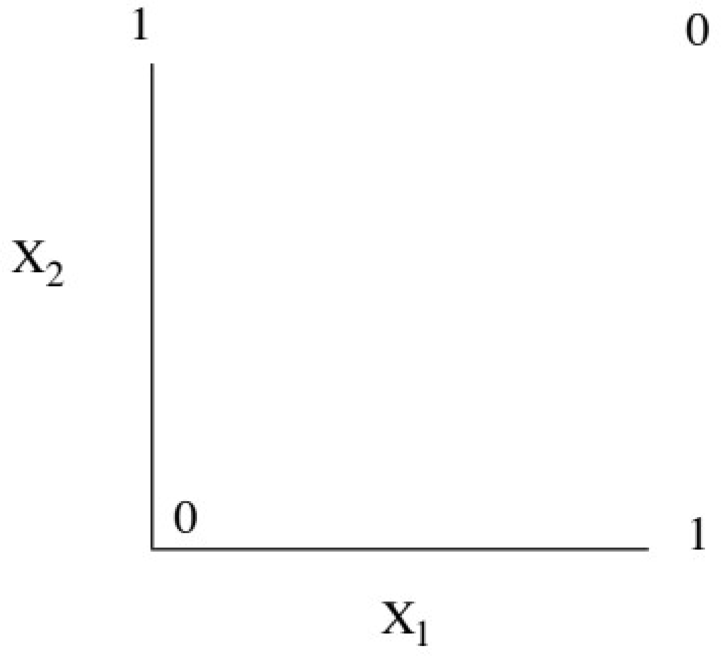
$$\Delta w_i = c(t - z)\, x_i$$

- Where w_i is the weight from input i to perceptron node, c is the learning rate, t is the target for the current instance, z is the current output, and x_i is i^{th} input
- Least perturbation principle
  - Only change weights if there is an error
  - small c rather than changing weights sufficient to make current pattern correct
  - Scale by x_i
- Create a perceptron node with n inputs
- Iteratively apply a pattern from the training set and apply the perceptron rule
- Each iteration through the training set is an *epoch*
- Continue training until total training set error ceases to improve
- Perceptron Convergence Theorem: Guaranteed to find a solution in finite time if a solution exists

How to Handle Multi-Class Output

- This is an issue with any learning model which only supports binary classification (perceptron, SVM, etc.)
- Create 1 perceptron for each output class, where the training set considers all other classes to be negative examples **(one vs the rest)**
  - Run all perceptrons on novel data and set the output to the class of the perceptron which outputs high
  - If there is a tie, choose the perceptron with the highest net value
- Create 1 perceptron for each pair of output classes, where the training set only contains examples from the 2 classes **(one vs one)**
  - Run all perceptrons on novel data and set the output to be the class with the most wins (votes) from the perceptrons
  - In case of a tie, use the net values to decide
  - Number of models grows by the square of the output classes

Exclusive Or



Is there a dividing hyperplane?